Lab 02

Random Wander and Avoid Obstacle

Reading: Ch. 3 of *Mobile Robotics for Multidisciplinary Study*, Berry

Read this entire lab procedure before starting the lab.

Purpose: An essential characteristic of an autonomous robot is the ability to navigate safely in an environment. The purpose of this lab is to develop random wander and obstacle avoidance behaviors for the GigaBot. The design of your program should use *subsumption architecture* where layer 0 of your control architecture will be the *collide* and *runaway* behaviors to keep the robot from hitting obstacles. Layer 1 will be the *random wander* behavior which moves the robot a random distance and/or heading every n seconds.

Objectives: At the conclusion of this lab, the student should be able to:

- Mount and wire infrared, lidar and sonar sensors on the robot
- Acquire and use data from all of the robot's range and distance sensors
- Write random wander, obstacle avoidance, and object follow behaviors on the GigaBot using sensor feedback with a bang-bang and proportional controller
- Use modular programming to implement subsumption architecture or a state machine on the GigaBot
- Move the robot safely in an environment with obstacles

Equipment:

base robot
4 infrared lidar sensors
2 sonar sensors
tape measurer

Theory:

Last week, you created the first two primitive motion behaviors on the robot: *Go-To-Goal* and *Go- To-Angle*. It is important to safely navigate a cluttered environment as you move to a goal in the world, so this week you will implement the *Avoid-Obstacle* and *Random Wander* behaviors. Each week, you will create more functionality and behaviors for your robot so you should design the system to be as modular as possible. This means try to re-use code and build on what you have done before. One way to do this is to use the subsumption architecture with layers where the most basic layer is motion control and obstacle avoidance. Subsumption architecture is a type of state machine where you turn layers on and off using inhibition and subsumption to create intelligent robot control.

Ultrasonic sensors emit a sound (ping) and measure the time of flight for the sound to return to calculate distance. *Lidar* sensors use pulsed light waves to emit a signal and uses time of flight for the light to determine how long it takes for the signal to return and calculate distance. The lidar sensor on your robot detects the presence or absence of an object. *Infrared* sensors return an analog value that is proportional to the distance to an object based upon time of flight for infrared light. Note that with IR, lidar, and sonar, the ambient light, color, texture, material, and angle of incidence determine how much energy is returned and this as well as specular reflection may affect accurate measurements.

Obstacle avoidance includes a *collide* and *runway* behavior will be based upon sensor feedback. The sensors are used to detect the presence or absence of an object or distance to an object. The error between the desired and actual distance will be used to control the robot motors or motion. The *collide* behavior will stop the robot if the distance to the obstacle is within a given error band, otherwise the robot continues to move forward. This type of control is BANG-BANG or ON-OFF control because either the controller turns on the robot's motors or not. The *runaway* behavior will use the sensor data to create a polar plot and the robot will move away proportional to the distance to the object as well as the direction of the obstacle. This type of control is called proportional control because the change in robot motion is affected by the magnitude of the error. Figure 1 shows an example of the feedback control system for robot obstacle avoidance behavior. Your job is to implement both of these on your mobile robot for obstacle avoidance.

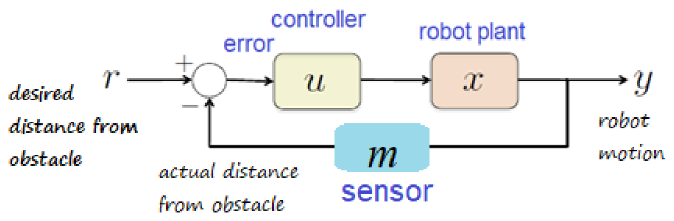


Figure 1: Obstacle Avoidance Feedback Control

Behavior-based programming uses primitive behaviors as modules for control. Primitive behaviors are concerned with achieving or maintaining a single, time-extended goal. They take inputs from sensors (or other behaviors) and send outputs to actuators (or other behaviors). Figure 2 is the Avoid-Obstacle behavior and Layer 0 of the subsumption architecture.

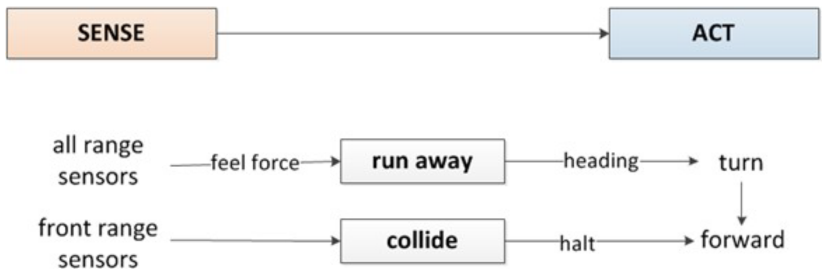


Figure 2: Level 0 – Obstacle Avoidance

To use the subsumption architecture to create a *smart wander* behavior (see Figure 3). The perceptual schema is *feelforce* and the motor schemas are *runaway* and *collide*. The primitive behaviors are also called *runaway* and *collide* and the two together make the abstract behavior, *obstacle avoidance*. The second layer of the architecture is the *wander* module. The wander module passes the heading to the *avoid* behavior which combines the feel force and wander heading to determine the direction the robot should turn to move away from obstacle. Note that the power of subsumption architecture is that output of the higher level subsumes the output from the lower level. The avoid module also can suppress the output from runaway and replace it to make the robot turn. Figure 3 is the full architecture for random wander and obstacle avoidance.

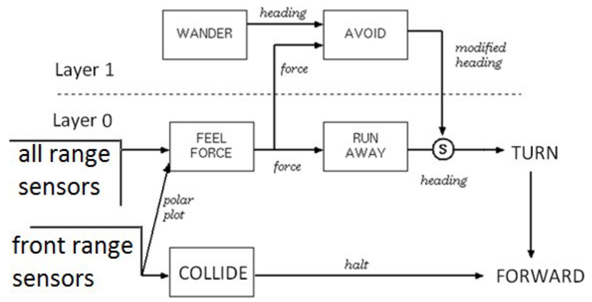


Figure 3: Smart Wander Behavior

To improve the random wander routine by integrating obstacle avoidance (collide and run away). The robot should wander randomly until an obstacle is encountered. The robot should *run away* from the obstacle and continue to wander. The robot's heading from the *wander* behavior should be modified based upon the force from the range sensors and then turn and move from the obstacle. The *avoid* module in Layer 1 combines the *FeelForce* vector with the *Wander* vector. Figure 4 shows an example of a *FeelForce* vector. The *Avoid* module then subsumes the heading from the Run Away module and replaces it with the modified heading as input to the *Turn* module. The power in this type of architecture is the flexibility in the execution based upon the use of inhibition and suppression. Figure 5 provides an example of the robot's sample motion for the Avoid-Obstacle behavior.

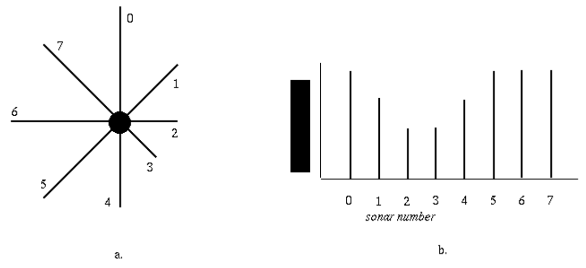


Figure 4: Robot Feel Force from sonar ring

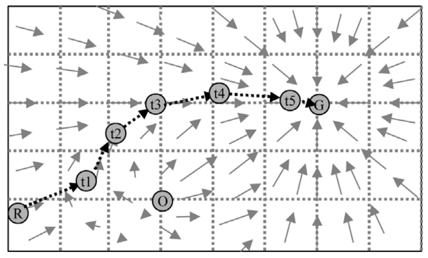


Figure 5: Potential Field Navigation,

References:

Use the following references to help you implement the various requirements in the lab procedure.

Sonar Sensor:	https://www.dfrobot.com/product-2172
Infrared Lidar Sensor:	https://www.pololu.com/product/4064
PID Tuning:	https://youtu.be/nvAQHSe-Ax4
Timers & Interrupts:	http://arduinoinfo.mywikis.net/wiki/Timers-Arduino#Arduino_Timers_and_Interrupts
GIGA R1 Wifi	https://docs.arduino.cc/hardware/giga-r1-wifi
GIGA R1 Dual Cores	https://docs.arduino.cc/tutorials/giga-r1-wifi/giga-dual-core
Interrupts:	http://playground.arduino.cc/Code/Interrupts
	http://playground.arduino.cc/Code/Timer1
Feedback Systems:	An Introduction for Scientists and Engineers, Karl J. Astrom and Richard M. Murray
	http://www.cds.caltech.edu/~murray/amwiki/index.php/Main_Page
	Tuning of a PID controller using Ziegler-Nichols Method
	https://www.scribd.com/document/233690247/Tuning-of-a-PID-controller-using-Ziegler-Nichols-Method

Pre-Lab:

- Create the pseudocode for the random wander behavior
- Create a flowchart for a proportional controller with sensor feedback for avoid obstacle behavior.
- Create a flowchart for a proportional controller with sensor feedback for follow object behavior.
- Create a state diagram for the smart wander behavior with 3 states including random wander, collide, and avoid obstacle. The inputs that transition the states are no obstacle, danger at obstacle, warning close to obstacle. Make sure to include a state transition table for the smart wander behavior state diagram.
- Create a state diagram for the smart follow behavior with 3 states including random wander, collide, and follow object. The inputs that transition the states are no obstacle, danger at obstacle, warning close to obstacle. Make sure to include a state transition table for the smart follow behavior state diagram.

Procedure:

Part I – Random Wander (Layer 1)

1. Make a copy of your Lab 01 code and rename it "***YourRobotName-Lab02.ino***".
2. Delete the Circle, Square, Figure 8 functions and any code and comments that are no long relevant to lab 02.
3. You will need to make a new header comment, new functions, new line comments, and new block comments for the lab 02 code.
4. Create a function in your code named randomWander(). There are no inputs or outputs.
5. It is your choice how to implement the random wander behavior on your robot but it should appear to move randomly in the environment.
6. In the function, generate a random number by using the following code. Your team should select the maximum number required based upon how you design the behavior.

```
maxVal = 1000;                //maximum value for random number
randomSeed(analogRead(0));    //generate a new random number each time called
randNumber = random(maxVal);  //generate a random number up to the maximum value
```

7. You decide how to use the random number to make the random wander function. For example, the robot can use this number for a turn angle, forward distance, steps to each motor separately, time to move. There are many ways to do this.
8. The green LED should be on when the robot is in random wander behavior.

Part II - Test the Sonar Sensors

1. ***BE CAREFUL! If you connect the sensor to the wrong power buss you could damage it! Or fry the microcontroller or damage your laptop.***
2. Connect the left sonar sensor in pin 4 on the Arduino GIGA. Make sure you are using the wire to connect between the row on the breadboard with the green wire on the sonar to pin 4.
3. Connect the right sonar sensor in pin 3 on the Arduino GIGA. Make sure you are using the wire to connect between the row on the breadboard with the green wire on the sonar to pin 3. See Figure 6.
4. Download the "***Sonar2024.ino***" code and read the comments to understand how it works.
5. Use your tape measure to test the output to determine the sonar range and dead zone for the left and right sensor. What are the reliable minimum and maximum readings? How does this compare to the datasheet?

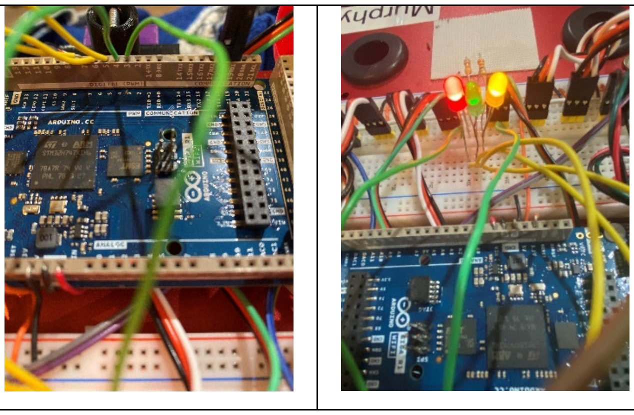


Figure 6: Wiring Sonar to GIGA

Part III – Test the Lidar Sensor

1. ***BE CAREFUL! If you connect the sensor to the wrong power buss you could damage it! Or fry the microcontroller or damage your laptop.***
2. Connect the left sonar sensor in pin 4 on the Arduino GIGA. Make sure you are using the wire to connect between the row on the breadboard with the green wire on the sonar to pin 4.
3. Connect the front, back, left, and right sensors to pins 8, 9,10,11, respectively. Make sure you are using the wire to connect between the row on the breadboard with the WHITE to the correct pin on the GIGA R1 WIFI microcontroller. See Figure 7.
4. Download the "***Lidar2024.ino***" code and read the comments to understand how it works.
5. Use your tape measure to test the output to determine the detect distance. It is an analog

output that indicates the distance to an object. How does this compare to the datasheet? Check the range and dead zone for all 4 sensors and put on the worksheet.

6. Think about how to use the dual core of the GIGA to read the sensor data in the background on M4 while running the main code on M7.

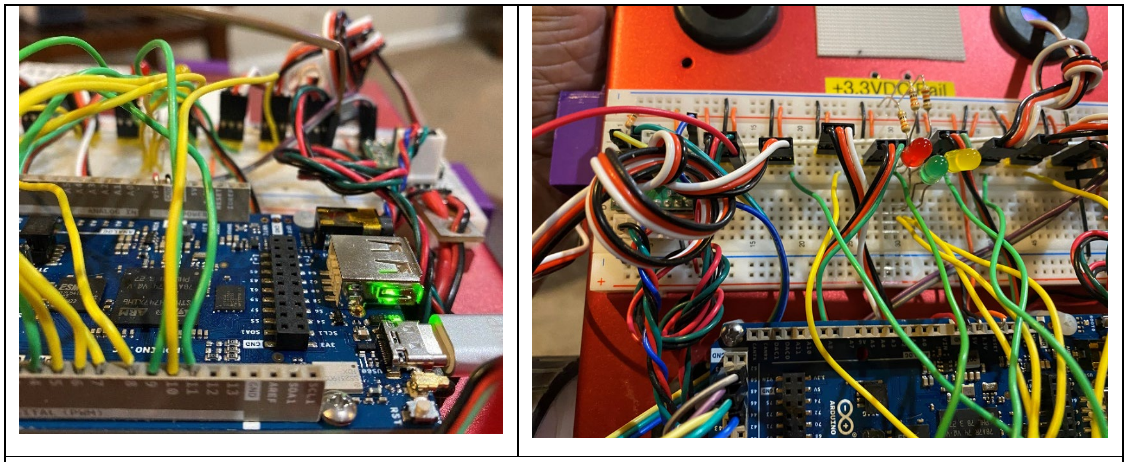


Figure 7: Wiring Lidar to GIGA

Part IV – Collide Behavior (Halt, Bang-Bang Control, Aggressive Kid)

In this lab, you will create two behaviors: *avoid-obstacle*, and *random wander*. The obstacle avoidance behavior will use either *collide and then run away* primitive behaviors. In the *collide* behavior, the robot will stop when an obstacle is detected and continue moving when the object is removed. To demonstrate the *collide* behavior, the robot should drive until it encounters an obstacle and stop without hitting it. Think of the collide behavior as the aggressive kid who comes close but then stops short from touching the object. Turn on the RED LED when the robot executes the aggressive kid behavior. Figure 8 shows an example of collide behavior with front sonar and lidar sensors. ***Read resources and learn how to use timer interrupts or the dual core to run the sensors in the background.***

Part V – Runaway Behavior (Potential Fields, Proportional Control, Shy Kid) (Layer 0)

In the *runaway* behavior, the robot will move forward or sit still and when an obstacle is detected, move away proportional to where the obstacle is felt (feel force). The robot's motion will be based upon the potential fields concept. For the *runaway* behavior, create a plot of the sensor readings and use the sum to create a repulsive vector to turn the robot away (i.e. *feel force*). The robot should turn by some angle proportional to the repulsive vector and continue moving. To demonstrate the *run away* behavior, the robot should sit in the middle of the floor until an object gets close and then move the opposite direction to get away based upon the potential field. Think of this behavior as the shy kid who does not want any object to get too close to him. It is possible to also show run away for an aggressive kid who moves forward and then moves away when an object gets close. It should be clear during the demonstration what type of behavior the robot is executing. Turn on the YELLOW LED when the robot executes the shy kid behavior. Figure 8 shows an example of runaway behavior.

Your program should provide a method to get the robot 'unstuck' if it approaches any local minima points (i.e. oscillates between two obstacles). Your program should be as modular as possible with multiple subroutines and behaviors that will be integrated in subsequent programs. Devise a method to test and confirm that your program works correctly and present the results in the laboratory memo.

Part VI – Follow Behavior (Potential Fields, Proportional Control, Curious Kid) (Alternate Layer 0)

In the *follow* behavior, the robot will move until an object is detected and then go into the halt (collide behavior and turn on the RED LED. Instead of executing the avoid behavior, it should use proportional control to track the object at a predetermine distance. The robot should speed up and slow down as necessary to stay within a certain distance of the object. Think of this as the curious kid behavior where the robot wants to follow the object to learn more. Turn on the RED and GREEN LED when this behavior is active on the robot. Figure 8 shows an example of follow behavior.

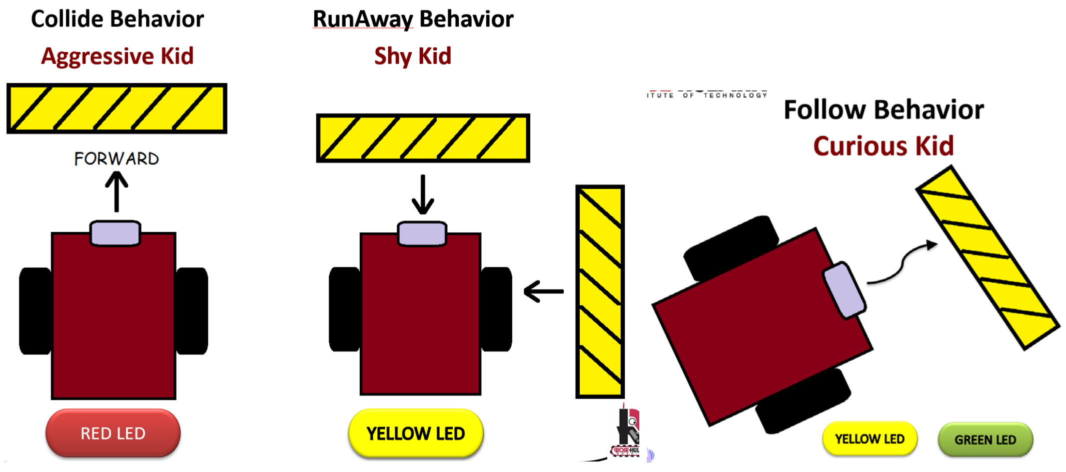


Figure 8: Collide, RunAWay, Follow Behaviors

Part VII – Smart Wander Behavior (State Machine, Subsumption Architecture)

Implement the full state machine on the robot. During the smart wander behavior, the robot should randomly move in the environment with the GREEN LED on. If an obstacle is detected within the danger distance the robot should execute the collide behavior and turn on the RED LED. The robot should then execute the avoid behavior and turn on the YELLOW LED. It should continue this behavior until the robot is clear of the obstacle.

Part VIII – Smart Follow Behavior (State Machine, Subsumption Architecture)

Implement the full state machine on the robot. During the smart wander behavior, the robot should randomly move in the environment with the GREEN LED on. If an obstacle is detected within the danger distance the robot should execute the collide behavior and turn on the RED LED. The robot should then execute the follow behavior and turn on the YELLOW and GREEN LEDs. It should continue this behavior until the robot loses the obstacle.

Submission Requirements:

Software Design Plan

For each lab you will submit a software design plan which may be in the form of pseudocode, a flowchart, state machine or subsumption architecture. The plan will be graded and you will get feedback on implementation before designing the full system.

Demonstration:

Bring your robot fully charged to class every day! Plug it in overnight.

During the demonstration, you will show each layer of the architecture separately.

- For layer 0, the robot should demonstrate shy kid and aggressive kid, separately. Please review the lab procedure if you don't recall what this means.
- For layer 1, to demonstrate random wander, the robot should turn at a random heading, forward and reverse direction periodically.
- To demonstrate the *Avoid* behavior, the robot should wander in the environment and halt when it "collides" with an obstacle, or modify the heading when it encounters and obstacle and then run away. The robot should give some type of audible and/or visual signal when an obstacle is encountered. Use RED, GREEN or YELLOW LEDs as described in the lab procedure.
- To demonstrate the *Follow* behavior, the robot should wander in the environment and halt when it "collides" with an obstacle, or modify the heading when it encounters and then follow at a desired distance. The robot should give some type of audible and/or visual signal when an obstacle is encountered. Use RED, GREEN or YELLOW LEDs as described in the lab procedure.

Program:

In subsequent weeks you will reuse this code thus your code should follow proper programming techniques such as detailed commenting and be as modular as possible where behaviors and reactive rules are separate functions.

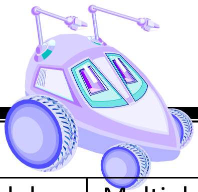
Use the following guidelines for creating your code.

- Code has a header comment with
- Program Name
  - Author Names
  - Date Created
  - Overall program description
  - Summary of key functions created with a description of each
- Code is modular with multiple functions called from the loop function. Functions have logical names that describe their functionality.
- Code is organized in a logical fashion with global and local variables with intuitive names.
- Code has block comments to introduce and describe each function and sufficient in line comments to describe how key parts of the code work.
- All functions and variables should have logical names that describe what they do or the value they hold. There should be no magic numbers or numeric values not related to a constant or variable to make it clear what it does.
- In general, someone unfamiliar with your project should be able to read your code and understand the functionality based upon your comments and naming convention.

Grading Rubric:

The lab is graded by the following rubric.

Points	Demonstration	Code	Memo
Excellent 100%	Excellent work, the robot performs exactly as required	Properly commented with a header and function comments, easy to follow with modular components	Follows all guidelines and answers all questions posed
Average 75%	Performs most of the functionality with minor failures	Partial comments and/or not modular with objects	Does not answer some questions and/or has spelling, grammatical, content errors



Poor 50%	Performs some of the functionality but with major failures or parts missing	No comments, not modular, not easy to follow	Multiple grammatical, format, content, spelling errors, questions not answered
Missing	Meets none of the design specifications or not submitted	Not submitted	Not submitted